



UNIVERSITÉ
DE MONTPELLIER



RAPPORT DE PROJET

Jeu Web : Désambiguisation lexicale



Groupe IMNA :
Isna OUAZI ,
Melissa MEKAOUI ,
Nicolas DELALANDE ,
Alexandre CULTY

Tuteur : Mathieu LAFOURCADE

TER M1
2016 - 2017

Table des matières

Introduction	4
1 Etat de l'art	5
1.1 Problématique et objectifs du projet	5
1.2 Les jeux sérieux	5
1.3 Les jeux sérieux dans le contexte du TALN	6
1.4 Caractérisation de l'ambiguïté lexicale	6
2 Analyse et conception	7
2.1 Étude préalable	7
2.1.1 Identification des acteurs	7
2.1.2 Besoins fonctionnels	7
2.2 Conception UML	8
2.2.1 Diagramme des cas d'utilisation	8
2.2.2 Diagrammes d'activités	11
2.2.3 Base de données	14
2.3 Choix relatifs au jeu	17
2.3.1 Mode de jeu	17
2.3.2 Points et crédits	17
2.3.3 Compétition	17
2.3.4 Problème lié aux joueurs	18
3 Réalisation	19
3.1 Environnement de développement	19
3.2 Prise en main de Symfony	19
3.2.1 Doctrine[3]	20
3.2.2 Bootstrap	23
3.3 Lancement d'une partie	23
3.4 Création d'une phrase	23
3.5 Édition de phrase	24
3.6 Export des données	26
3.7 Questions d'optimisation	27
3.8 Tests	27
4 Déploiement	28
4.1 Caractéristiques de l'hébergement	28
4.2 Configuration	28
4.3 Base de données	29
4.4 Problèmes rencontrés	29
5 Résultats et perspectives	30
5.1 Résultats et retours utilisateurs	30
5.2 Perspectives	30
6 Conclusion	32
Annexes	34

Table des figures

2.1	Diagramme des cas d'utilisation	8
2.2	Jouer une partie : Diagramme d'activité	11
2.3	Créer une phrase : Diagramme d'activité	12
2.4	Modérer une phrase : Diagramme d'activité	13
2.5	User Bundle	14
2.6	Ambiguss Bundle	15
2.7	Judgment et App Bundle	16
3.1	Liaison auteur dans MotAmbigu	21
3.2	Liaison gloses dans MotAmbigu	21
3.3	Utilisation du QueryBuilder	22
3.4	Exemple de phrase à la création	23
3.5	Exemple de phrase	24
3.6	Exemple d'export de données	26
6.1	Génération d'une entité via la console	34
6.2	Entité générée	35
6.3	Utilisation d'une entité	36
6.4	Jouer une phrase	39
6.5	Résultat	39
6.6	Jeu sur mobile	40
6.7	Création d'une phrase	40
6.8	Page d'édition de phrases	41
6.9	Classement des joueurs par points	41

Remerciements

Nous tenons tout d'abord à remercier notre tuteur de projet, Monsieur Mathieu Lafourcade, qui nous a permis de réaliser ce projet dans les meilleures conditions, notamment grâce à ses conseils et sa disponibilité tout au long de ce semestre.

Nous adressons aussi nos sincères remerciements à Madame Le Brun, Monsieur Lafourcade et tous les autres testeurs du jeu pour le temps qu'ils ont accordé à notre projet afin de nous faire parvenir leurs avis et conseils.

Introduction

Dans le cadre de notre formation en première année de master informatique, nous sommes amenés à réaliser un travail d'étude et de recherche (TER) en groupe sur l'un des nombreux projets proposés par les enseignants du département informatique de l'université de Montpellier.

Nous réaliserons dans le cadre de ce TER un projet intitulé « Jeu web d'annotation de textes pour la désambiguïsation lexicale ». Ce projet aura pour but de réaliser un jeu de type GWAP (game with a purpose), jouable sur web et sur mobile, dont les données récoltées (ressources lexicales) serviront au traitement automatique du langage naturel (TALN).

Concrètement, notre jeu permettra de produire un corpus de phrases/textes annotés sémantiquement. Chaque terme ambigu (ex : livre) sera associé à une ou plusieurs gloses permettant d'identifier le sens correct (poids, objet, monnaie, verbe...). Le jeu collectera l'ensemble des réponses des joueurs et distribuera des points en fonction de l'adéquation de celles-ci. Afin de rendre le jeu addictif, plusieurs mécanismes de gamification ont été implémentés.

Ce document sera divisé en quatre grands chapitres : une partie état de l'art qui comportera des définitions concernant les GWAP et le « serious game », une partie dédiée à la conception et la modélisation du jeu (mise en place du cahier des charges, diagrammes UML...), une autre à son développement et enfin une partie consacrée aux résultats obtenus après l'intégration du jeu et de sa mise en ligne.

1. Etat de l'art

1.1 Problématique et objectifs du projet

Ce projet avait pour but de réaliser un jeu de type GWAP jouable à la fois sur ordinateur et sur mobile, et dont les données récoltées serviraient au traitement automatique du langage naturel. Ce jeu permettra donc de produire un corpus de phrases/textes annotés sémantiquement où chaque terme ambigu (exemple : livre) sera associé à une ou plusieurs gloses permettant d'identifier son sens correct (poids, objet, monnaie, verbe...).

Avec ces objectifs, nous avons défini les problématiques suivantes :

- Comment développer un jeu utilisable à la fois sur ordinateur et sur mobile ?
- Comment rendre le jeu attractif ?

1.2 Les jeux sérieux

Selon une définition de 2008 [6] : « la vocation d'un serious game est d'inviter l'utilisateur à interagir avec une application informatique dont l'intention est de combiner à la fois des aspects d'enseignement et d'apprentissage avec des ressorts ludiques issus du jeu vidéo. De manière plus simple c'est donner à un contenu utilitaire (serious), une approche vidéoludique (game) ».

Cette définition a quelque peu évolué depuis que les "développeurs" ont pris conscience que les utilisateurs pouvaient eux aussi contribuer à la science. Désormais un jeu sérieux possédant un but (Game with a purpose abrégé en GWAP) « est un jeu à travers lequel l'activité ludique permet de collecter des données ou de résoudre des problèmes trop complexes, ou trop coûteux en termes de moyens humains et matériels pour être résolus par des machines » [7].

Aujourd'hui, nous retrouvons les gwaps dans divers secteurs d'activités tels que la biologie ou la médecine (Phylo, EyeWire, NanoDoc...) mais également dans un domaine qui nous intéresse particulièrement dans le cadre de ce projet : le traitement automatique du langage naturel (TALN).

1.3 Les jeux sérieux dans le contexte du TALN

Dans le contexte qui nous intéresse, les jeux sérieux ont vocation à créer des ressources lexicales, voici quelques exemples de ressources :

- associations entre un mot et ses synonymes (maison et bâtiment)
- relations entre deux mots (feuille avec plante)
- des traductions d'un mot (vert et green)
- différents sens d'un mot (avocat : fruit ou métier)

Bien que ces relations et associations entre mots nous semblent évidentes, ces ressources sont difficiles à créer : le faire de façon automatisée se heurte au problème de la sémantique des mots, et les élaborer manuellement est long, coûteux et fastidieux. Dans ce contexte, l'utilisation des jeux sérieux est donc très pertinente.

Ces ressources sont utiles dans divers contextes de TALN, dont la traduction automatique, l'extraction d'informations dans un texte, la désambiguïsation d'une phrase etc...

1.4 Caractérisation de l'ambiguïté lexicale

«L'ambiguïté est la propriété d'un mot, d'une suite de mots ou d'un concept ayant plusieurs significations ou plusieurs analyses grammaticales possibles.» [1]

En d'autres termes, un mot ou une expression sont lexicalement ambigus si on peut leur faire correspondre au moins deux sens différents. Par exemple, dans la phrase "L'avocat a bien défendu son client.", l'avocat ici est le plaideur, non le fruit, et on le déduit très simplement à l'aide du contexte.

2. Analyse et conception

2.1 Étude préalable

L'étude préalable constituant une étape préliminaire à la réalisation d'une application, nous avons effectué en premier lieu une analyse des besoins fonctionnels du jeu en commençant par définir les acteurs du jeu puis une modélisation UML¹ sera présentée pour mieux expliquer son fonctionnement.

2.1.1 Identification des acteurs

- **Visiteur** : Un visiteur est un joueur anonyme , il ne possède pas de compte dans le jeu mais il peut quand même y jouer. Cependant il ne récoltera aucun point et ne pourra pas accéder aux autres fonctionnalités du jeu. Ses réponses ne sont pas prises en compte pour éviter les abus.
- **Membre** : Un membre est un joueur possédant un compte dans le jeu, il peut ainsi jouer, créer des phrases, se voir attribuer un titre, dépenser ses crédits...
- **Modérateur** : Un joueur tenant le rôle de modérateur est un membre actif du jeu. Son statut lui permet entre autres d'avoir accès aux objets signalés (phrases, mots ambigus et gloses) et de les corriger.
- **Administrateur** : L'administrateur, quant à lui, est chargé du maintien et de l'évolution du jeu. Il aura accès aux pages de modération, aux statistiques de jeu et à la base de données. C'est également à lui que revient la tâche de nommer les modérateurs.

2.1.2 Besoins fonctionnels

Nous avons identifié les besoins suivants lors de notre analyse, classés par ordre de priorité :

- Faire en sorte que l'interface du site soit ergonomique, aussi bien sur ordinateur que sur téléphone portable ou tablette afin de pouvoir jouer n'importe où.
- Pouvoir créer des phrases, des mots ambigus et des gloses.
- Pouvoir jouer sans que les phrases ne se répètent.
- Créer des classements (utilisateurs, phrases..) pour rendre le jeu plus addictif.
- Pouvoir consulter les profils des utilisateurs.
- Récupérer les données du jeu et les formater afin de les rendre facilement utilisables dans d'autres projets.
- Pouvoir "aimer" et partager une phrase sur les réseaux sociaux.

1. Unified Modeling Language

- Pouvoir signaler un objet (phrase, mot ambigu, glose) en cas de faute, ou d’abus.
- Corriger ou modifier les objets signalés.

D’autre part le projet avait vocation à être jouable le plus tôt possible car il était important de recueillir les avis de plusieurs bêta-testeurs pour éventuellement modifier certains aspects du jeu au cours du projet puisque nous utilisons une méthode de gestion de projet Agile.

2.2 Conception UML

Nous commencerons par présenter le diagramme de cas d’utilisation qui illustrera les fonctionnalités que propose le jeu, puis les diagrammes d’activités pour mieux expliquer le déroulement de ces fonctionnalités dans le jeu.

2.2.1 Diagramme des cas d’utilisation

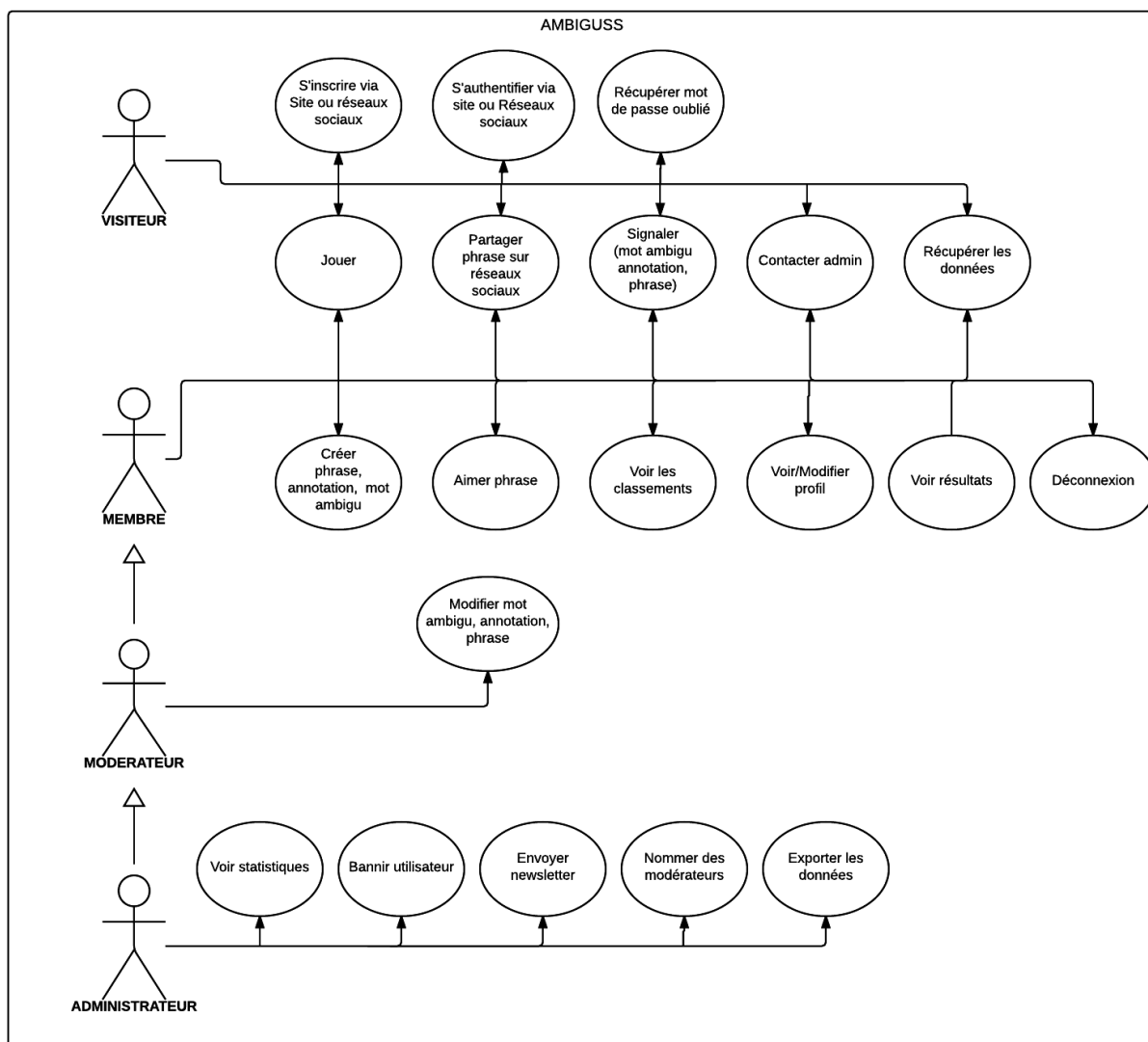


FIGURE 2.1 – Diagramme des cas d’utilisation

Description des cas d'utilisation

Note 1 : la colonne utilisateurs représente le rôle minimum à avoir afin d'accéder à la fonctionnalité. Ainsi «visiteur» représente un utilisateur non connecté et «membre» un utilisateur connecté. Un modérateur est aussi un membre, et un administrateur est à la fois modérateur et membre. Les contraintes sont au moins de respecter le type d'utilisateur de la fonctionnalité, plus celles indiquées.

Note 2 : le tableau représente les cas d'utilisations les plus importants, un tableau comportant tous les cas se trouve en annexes.

Cas d'utilisation	Description	Utilisateurs	Contraintes
Jouer	En cliquant sur le bouton « jouer », Une phrases tirée au hasard est affichée. Pour jouer l'utilisateur devra associer la bonne glose à chaque mot ambigu (affiché en rouge) La page des résultats est ensuite affichée avec les points obtenus ainsi que les réponses des autres joueurs. Il est également possible de “passer” une phrase, cela lancera une nouvelle partie.	Visiteur, Membre	
Signaler (mot ambigu, annotation, phrase)	Toute phrase, annotation ou mot ambigu que l'utilisateur jugera inappropriés ou faux pourront être signalés en remplissant un petit formulaire avec la catégorie du signalement(faute d'orthographe, insultes, trolls...) et une description du problème.	Membre	-Remplir correctement le formulaire.
Créer (phrase, annotations, mots ambigus)	Créer une phrase revient à créer une partie. Il sera demandé à l'utilisateur de définir les mots ambigus de la phrase en cliquant sur le bouton «Rendre le mot ambigu » et d'y associer des annotations (2 minimum). Il est également possible de créer des annotations autrement que lorsqu'on crée une partie en cliquant sur le bouton « ajouter une glose » lorsqu'on joue et que les gloses ne nous satisfont pas.	Membre	-Avoir assez de crédits.

Cas d'utilisation	Description	Utilisateurs	Contraintes
Récupérer les données	Toutes les réponses des utilisateurs sont enregistrées dans la base de données puis traitées afin de les mettre à disposition des utilisateurs (fichier au format json téléchargeable)	Visiteur, Membre	
Modifier mot ambigu, annotation, phrase	Lorsqu'une phrase ou une annotation est signalée, elle est immédiatement affichée dans un tableau que seuls les modérateurs peuvent voir. De là ils auront la possibilité d'agir en corrigeant la phrase, le mot ambigu ou l'annotation ou en les supprimant si le contenu est inapproprié.	Modérateur	

2.2.2 Diagrammes d'activités

Ces diagrammes décrivent les messages échangés entre l'utilisateur et le système pour mieux entrevoir le déroulement des fonctionnalités les plus importantes du jeu :

— Jouer une partie

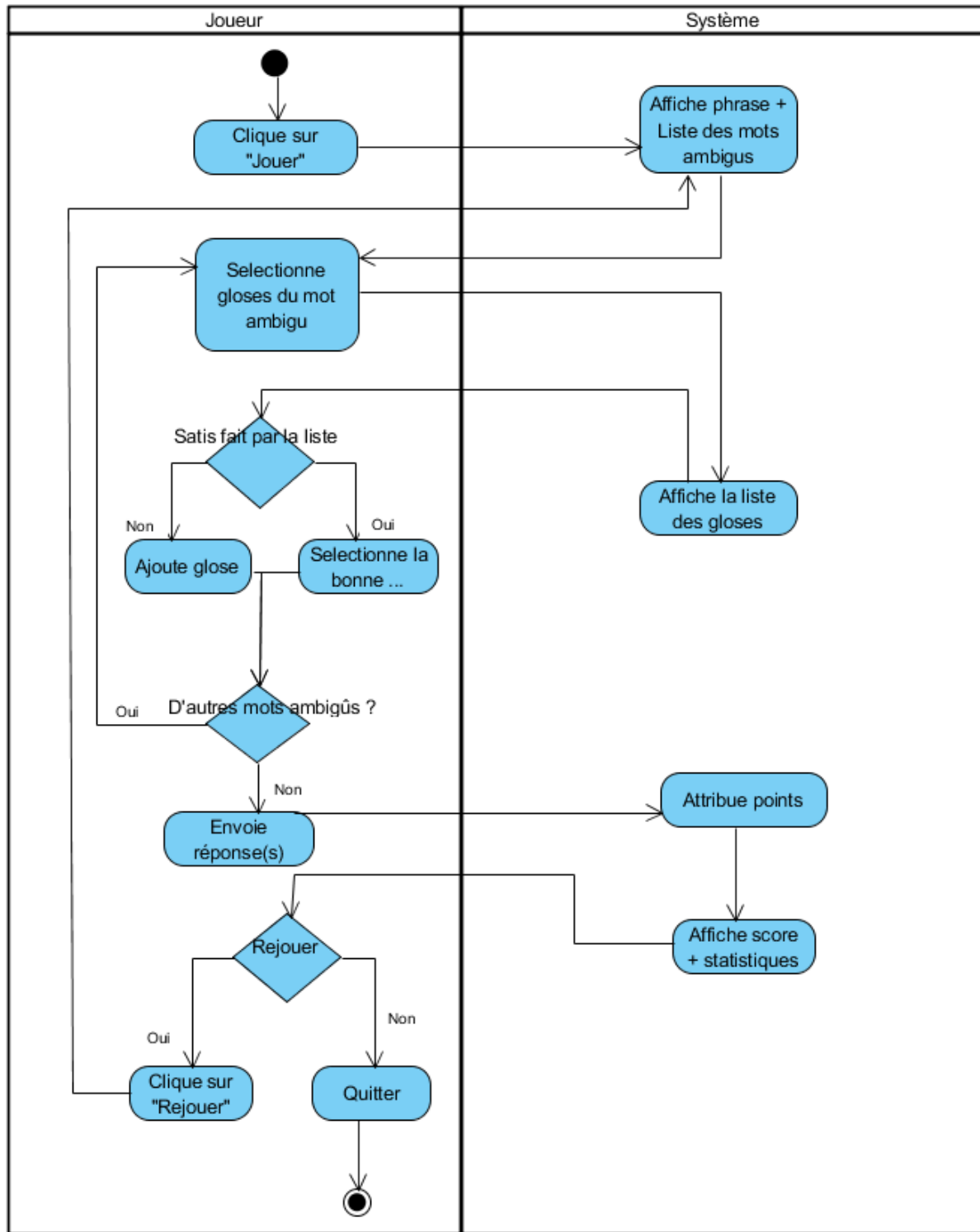


FIGURE 2.2 – Jouer une partie : Diagramme d'activité

— Créer une phrase

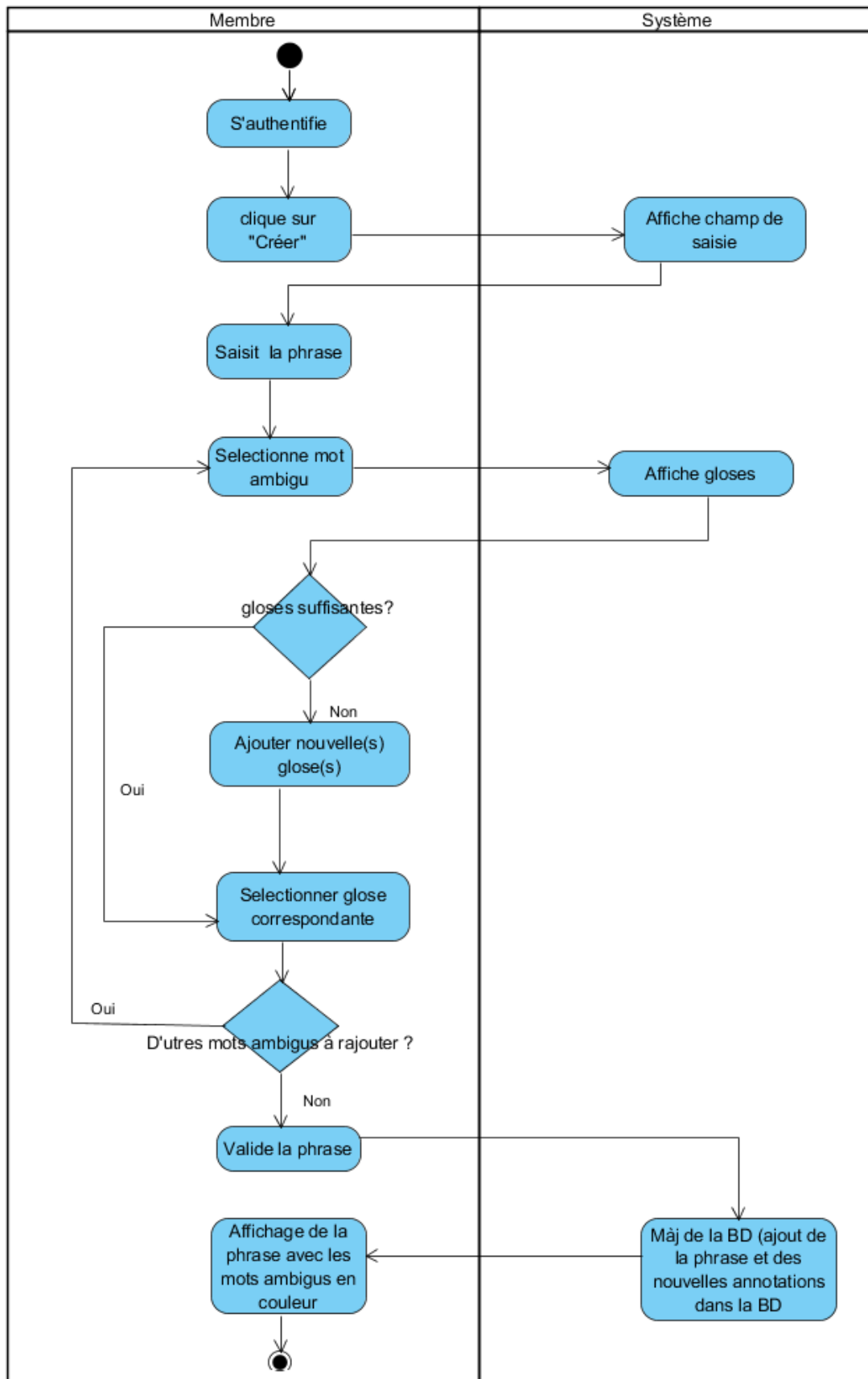


FIGURE 2.3 – Créer une phrase : Diagramme d'activité

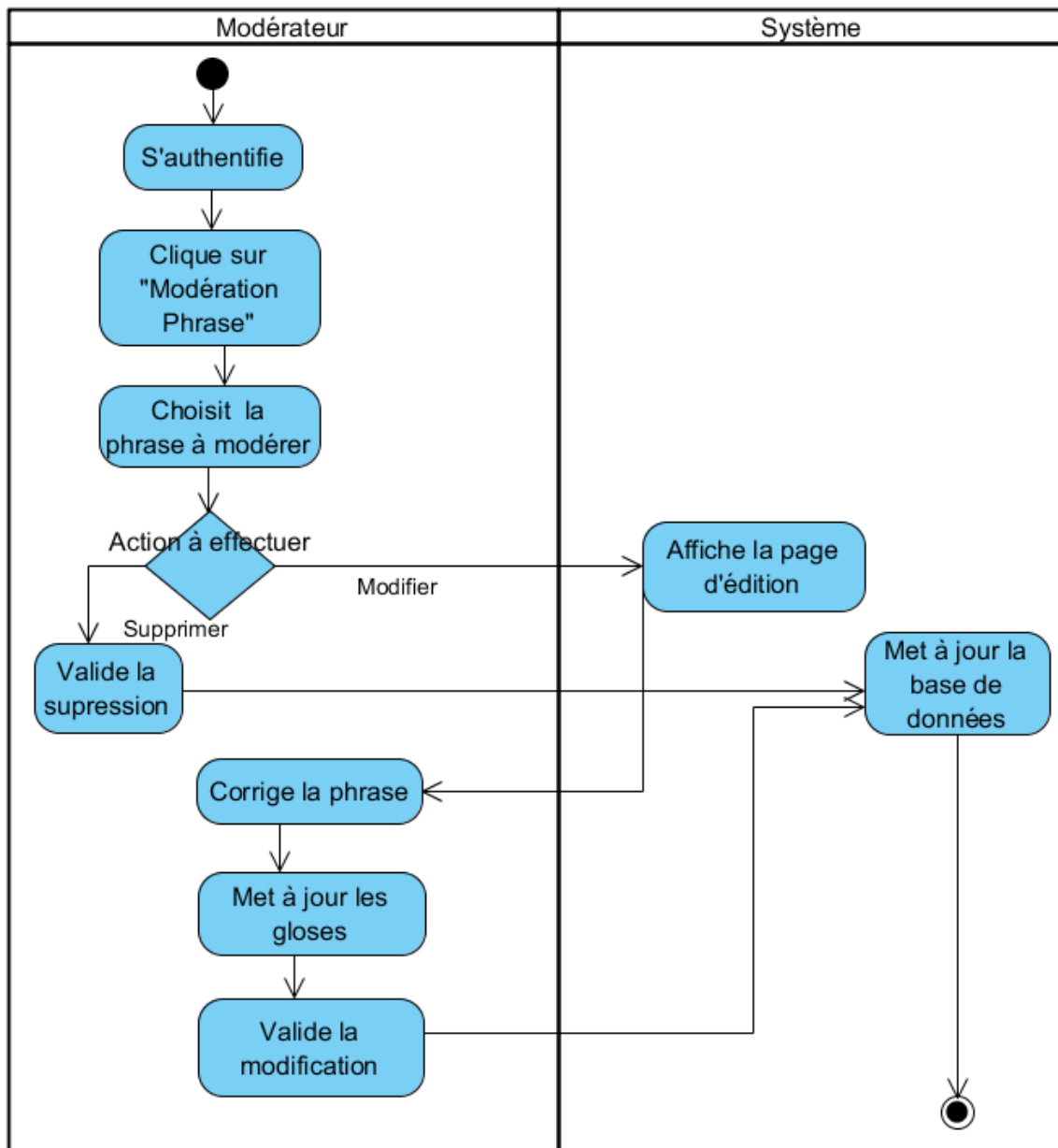


FIGURE 2.4 – Modérer une phrase : Diagramme d'activité

2.2.3 Base de données

Après avoir modélisé les fonctionnalités premières du jeu, nous avons mis en place un diagramme de classes sur lequel nous nous sommes basés lors de la mise en place de la base de données de l'application.

Nous avons pris soin de découper ce diagramme en séparant les données du jeu, les données relatives aux joueurs et les données relatives à la modération.

Dans la figure suivante nous voyons le User Bundle² dont la classe la plus importante est Membre qui contient toutes les informations concernant un joueur.

Un membre est associé à un groupe qui dispose des droits de ce groupe, mais peut aussi avoir des rôles particuliers. Il est aussi relié à un historique des actions qui le concernent.

Actuellement non utilisé, les succès peuvent être acquis par un membre dans une optique de récompense, et d'augmentation de l'attrait du jeu.

Le reste des liaisons servent surtout à l'identification, par exemple : qui a envoyé une newsletter, qui a créé une phrase, un signalement en question etc...

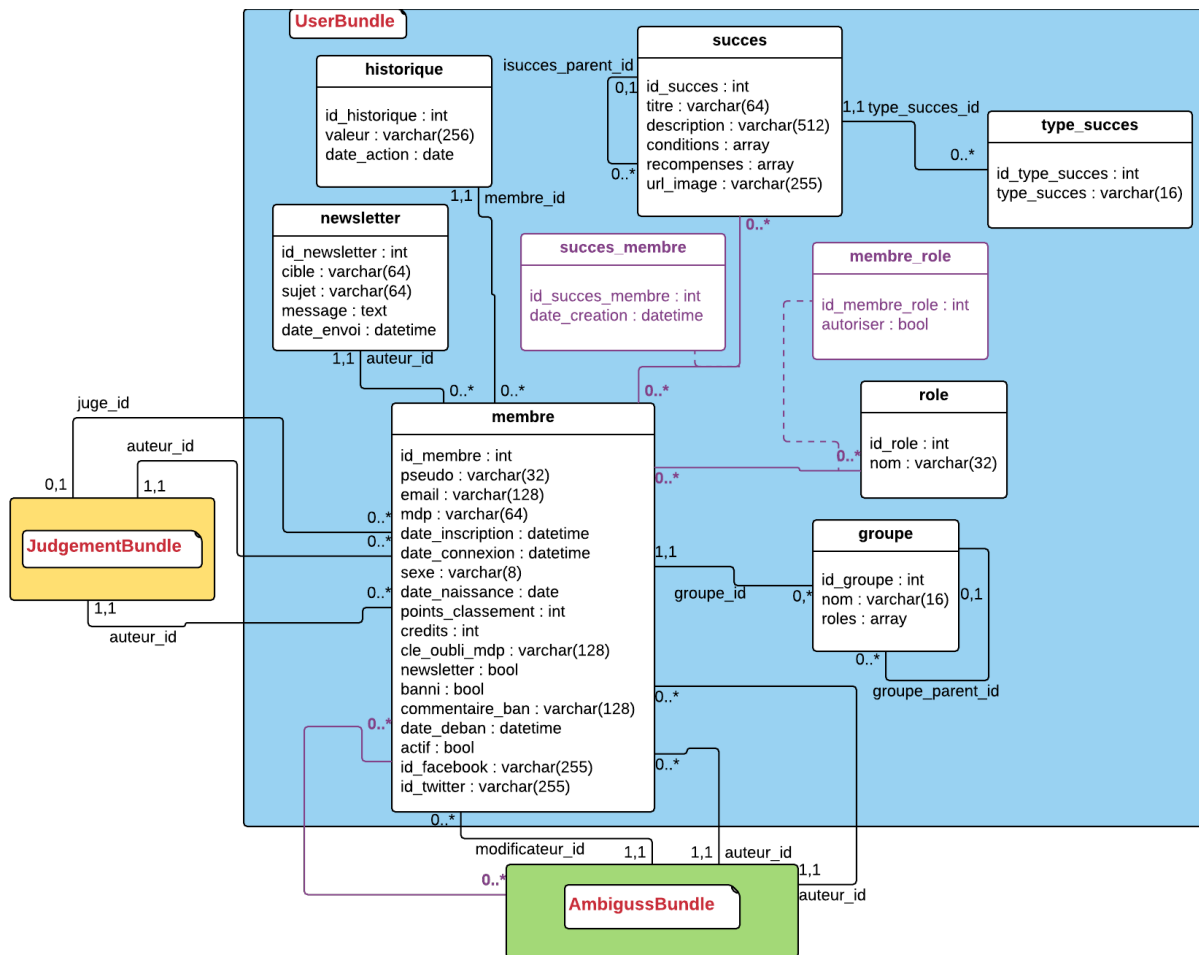


FIGURE 2.5 – User Bundle

2. Un bundle est l'équivalent d'un package, cela permet de séparer un projet en plusieurs parties

Dans la figure suivante nous pouvons observer le bundle Ambiguss. Il rassemble toutes les classes relatives au fonctionnement du jeu. C'est à partir de celui-ci que l'on pourra extraire les données intéressantes du jeu.

Une phrase peut être reliée à plusieurs mots ambigus qui eux-même peuvent se trouver dans différentes phrases. De même les mots ambigus sont reliés à plusieurs gloses pouvant les décrire et certaines gloses peuvent être communes à plusieurs mots ambigus.

Une réponse quant à elle, associe une glose à un mot ambigu dans une phrase spécifique.

La classe poids_reponse permet à un joueur d'indiquer s'il considère une glose comme étant juste, fausse ou s'il ne sait pas. Cette classe a été imaginée pour un mode de jeu différent où le joueur devait plutôt indiquer pour chaque glose liée au mot ambigu s'il la trouvait juste ou fausse. Ce mode compliquait le jeu tout en le rendant fastidieux dans le cas où un mot serait associé à un grand nombre de gloses. Avec le mode de jeu actuel, toutes les réponses sont considérées comme justes, mais après discussion avec notre tuteur, il a été décidé de garder la classe au cas où.

Nous remarquons aussi la classe aimer_phrase qui enregistre les "j'aime" d'une phrase, et la table partie qui elle enregistre les phrases jouées par un membre.

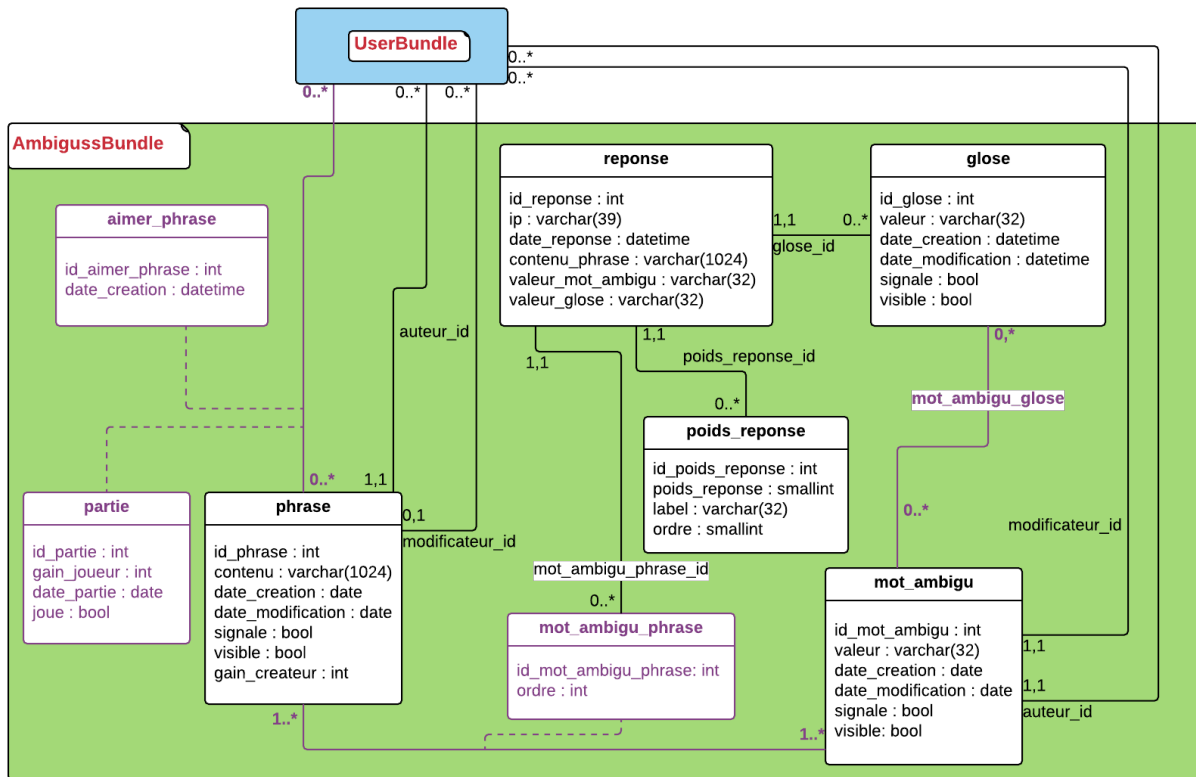


FIGURE 2.6 – Ambiguss Bundle

Dans la figure suivante nous nous intéressons aux deux derniers bundles.

Le premier, App, contient seulement la classe visite qui permet de collecter des informations au sujet des visiteurs non enregistrés.

Le deuxième concerne les mécanismes de signalement. À la base nous avons envisagé une sorte de modération participative où chaque joueur pourrait voter pour ou contre les changements proposés. Cependant la modification des données est quelque chose de complexe car comme on l'a vu précédemment toutes les classes d'Ambiguss Bundle sont étroitement liées. Cette difficulté est détaillée plus loin mais c'est la raison pour laquelle actuellement seuls les modérateurs peuvent effectuer des corrections.

La classe vote_jugement n'est actuellement pas utilisée car nous n'avons pas eu le temps de développer cette partie.

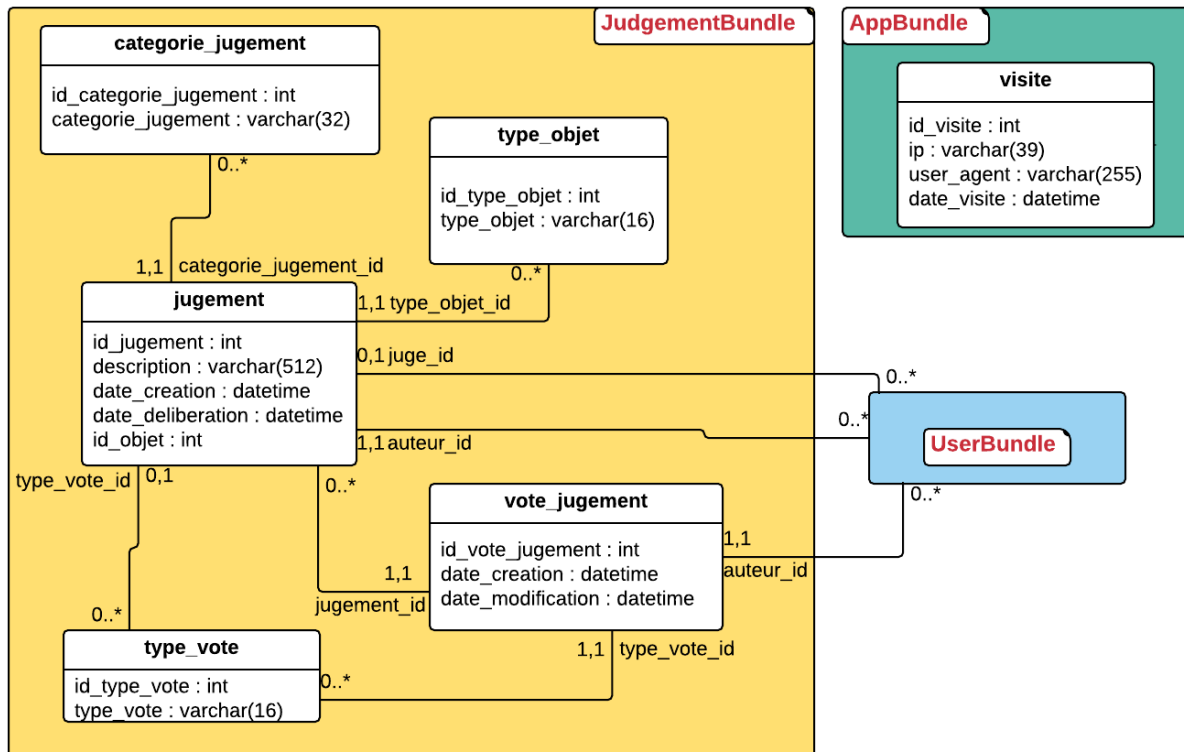


FIGURE 2.7 – Judgment et App Bundle

2.3 Choix relatifs au jeu

2.3.1 Mode de jeu

Le concept du jeu est simple : une phrase dans laquelle un ou plusieurs mots sont ambigus est proposée au joueur. Celui-ci doit alors indiquer pour chacun de ces mots ambigus la glose qui lui paraît la plus pertinente afin de lever l'ambiguïté.

Initialement, le jeu devait comporter trois niveaux de difficulté. Un niveau "facile" où le joueur se voyait proposer plusieurs gloses pour chaque mot ambigu. Un niveau "moyen" où les mots ambigus étaient signalés comme tels, mais où aucune glose n'était proposée. Puis un niveau "difficile" où rien n'était indiqué : le joueur devait lui-même décider quels mots étaient ambigus de son point de vue, et quelles gloses y associer.

Grâce aux retours de nos bêta-testeurs nous nous sommes rendu compte que les modes "moyen" et "difficile" seraient certainement moins amusants et moins addictifs que le mode facile. De plus ils risqueraient d'engendrer une grande quantité de gloses, en effet dans les modes de jeu où les gloses ne sont pas suggérées les joueurs risquent d'ajouter, sans le savoir, des gloses très similaires à celles existantes. Ces doublons nuisent ensuite à l'expérience des joueurs confrontés à une longue liste de gloses, ils empêchent également l'émergence d'un consensus clair autour d'un ou plusieurs sens par la dispersion des votes. Ces modes de jeu n'ont donc pas été développés.

2.3.2 Points et crédits

Nous voulions que les joueurs réalisent en priorité deux actions sur le jeu : jouer et créer des phrases. De plus, nous voulions que les joueurs répondent de leur mieux afin d'obtenir des données de qualité. De même, il fallait encourager les joueurs à créer des phrases correctes et intéressantes.

Il existe deux types de monnaie que les joueurs peuvent acquérir de différentes façons, notamment en jouant mais aussi en créant des phrases. D'une part les "points", qui ne peuvent pas être dépensés, ils servent aux joueurs à se mesurer entre eux afin de créer un esprit de compétition et donc inciter les joueurs à jouer toujours plus. D'autre part, il y a les "crédits" qui peuvent être dépensés pour créer des phrases et ajouter des gloses.

Ces points et ces crédits sont gagnés quand un joueur répond à une phrase. Le montant varie en fonction de l'exactitude de la réponse : plus la réponse choisie correspond à la majorité des précédentes réponses plus elle est considérée comme exacte et donc plus elle rapporte.

Les joueurs peuvent également "aimer" une phrase, cette action rapporte une petite somme de crédits au créateur de la phrase. De plus, à chaque fois qu'un joueur joue une phrase, le créateur de la phrase gagne 10% des crédits générés. Il y a donc un vrai intérêt à créer des phrases intéressantes que les autres auront envie de "liker" et d'essayer.

2.3.3 Compétition

La compétition entre les joueurs crée une émulation qui est propice à l'addiction et donne envie de rejouer régulièrement. Il était essentiel pour notre jeu d'utiliser ce concept afin que les joueurs ne se lassent pas trop vite, nous avons donc créé plusieurs classements pour provoquer ce sentiment de concurrence.

Tout d'abord un classement général qui présente les statistiques globales classées selon le nombre de points. Dans ces statistiques on trouve : le nombre de points accumulés, le nombre de phrases créées et le nombre total de "like" que les utilisateurs ont distribués.

Un titre honorifique est aussi attribué en fonction du classement, ainsi le joueur avec le plus de points se voit attribuer le titre d'Ambigator. Là encore le but est de faire naître une compétition entre les joueurs afin de les faire jouer plus.

Les joueurs peuvent néanmoins organiser le classement en fonction d'attributs secondaires, pour voir qui a créé le plus de phrases par exemple.

Nous avons aussi créé un classement spécifique aux phrases. On peut y voir les phrases qui ont eu le plus de "like" ou qui ont rapporté le plus de points à leur créateur. Il est d'ailleurs possible de cliquer sur une phrase de ce classement afin de la jouer. Les joueurs peuvent ainsi jouer sur les phrases qu'ils veulent.

Enfin, le dernier classement est celui des phrases créées par le joueur. Il y a les mêmes informations que dans le précédent classement.

2.3.4 Problème lié aux joueurs

Dans tous les GWAP il y a le risque que des joueurs introduisent des données fausses, que cela soit de manière intentionnelle ou pas. Il faut donc que le jeu soit résilient par rapport à cette possibilité.

Un des risques majeurs serait qu'un joueur choisisse sciemment les mauvaises gloses. Pour limiter l'impact d'un tel agissement, nous avons décidé qu'une réponse ne serait comptabilisée que la première fois. Ainsi le problème se résout tout seul si le nombre de joueurs est assez élevé car l'avis de la majorité compensera largement les quelques mauvais choix. De plus les votes des visiteurs, c'est-à-dire les gens qui jouent sans être inscrits, ne sont pas comptabilisés. Néanmoins, le risque existe que sur une phrase difficile, la majorité des gens échouent de bonne foi. Mais ce risque est acceptable et offre finalement une information intéressante.

Un autre risque serait qu'un joueur crée un grand nombre de phrases n'ayant aucun sens ou véhiculant des idées ou propos condamnables (racisme, etc...). Cependant la création d'une phrase coûte des crédits, et il faut jouer sérieusement pour en obtenir une quantité suffisante. Un joueur malintentionné se retrouve face à un dilemme : soit il joue mal et ne pourra pas créer de phrases, soit il joue bien et pourra créer les phrases qu'il souhaite. Dans ce dernier cas le joueur contribue à la cohérence et à la qualité des réponses, et les phrases de mauvais goût qu'il pourrait créer seront rapidement signalées par d'autres joueurs puis supprimées par les modérateurs. De plus un joueur peut être banni si son comportement est vraiment néfaste pour le jeu.

L'ajout de nouvelles gloses sur un mot ambigu pose les mêmes problèmes. Ainsi pour éviter qu'un joueur ne rentre des centaines de gloses inappropriées, nous avons aussi affecté à cette action un coût en crédits.

La possibilité de signaler une phrase permet non seulement de retirer les phrases problématiques comme vu précédemment, mais elle permet également de corriger des petites erreurs laissées par les joueurs de bonne foi, comme les fautes d'orthographe par exemple. Mais pour limiter le travail des modérateurs, les joueurs peuvent corriger leurs propres phrases au cours des 5 minutes qui suivent leur création car celles-ci ne sont pas immédiatement proposées aux autres joueurs.

3. Réalisation

3.1 Environnement de développement

Chacun était libre de choisir son environnement de développement mais plusieurs d'entre nous avons choisi PhpStorm, logiciel d'édition PHP développé par JetBrains, société notamment connue pour IntelliJ, son logiciel d'édition Java. Le logiciel est normalement payant (environ 199€/an), mais en tant qu'étudiants, nous pouvons bénéficier d'une licence gratuite. En plus de la coloration syntaxique quasiment obligatoire pour n'importe quel logiciel d'édition aujourd'hui, il propose une auto-complétion du code et intègre des outils pour gérer son projet via Git, ce qui le rend très pratique pour nous. De plus, des plugins sont mis à disposition pour améliorer les conditions de développement, notamment un plugin pour Symfony dont nous nous sommes servis, qui permet d'auto-compléter les méthodes et configurations du framework qui est vraiment très pratique.

Nous avons dès le début du projet configuré nos environnement de la même façon afin d'être tous dans les mêmes conditions de développement. Pour cela nous avons modifié nos fichiers hosts afin que l'adresse URL `ambiguuss.calyxe.dev` redirige vers la boucle locale (`127.0.0.1 / localhost`), et configuré des hôtes virtuels avec Apache afin que les requêtes sur `ambiguuss.calyxe.dev` soit redirigées sur le répertoire de notre système.

3.2 Prise en main de Symfony

Symfony est un framework¹ PHP français. Développée par la société SensioLabs, la première version est parue en octobre 2005. Aujourd'hui certains composants de Symfony sont utilisés par de grands sites comme Yahoo, Dailymotion, mais aussi par de gros CMS (Content Management System, système de gestion de contenu) comme Drupal, phpBB, eZPublish, Joomla, Magento, et même d'autres frameworks comme Laravel.

Aucun membre du groupe n'avait déjà vraiment utilisé Symfony, ce fut donc une découverte totale pour nous quatre. Pour apprendre à nous servir de Symfony, nous avons tous suivi le tutoriel de son créateur (Fabien Potentier) disponible sur OpenClassRooms[4] ainsi que la documentation sur les sites officiels des divers composants.

Parmi les composants et outils les plus importants nous trouvons :

- **Twig** : moteur de templates. Son objectif est de séparer le code PHP du code HTML. On peut noter qu'il possède une notion d'héritage entre les vues qui évite la duplication de code.[5]
- **Doctrine** : ORM (Object-Relational Mapping). Son but est de s'occuper de l'enregistrement des données et d'avoir une approche objet
- **Les contrôleurs** : contiennent la logique de l'application.

1. Framework : ensemble d'outils servant de base pour le développement d'une application

- **Les routeurs** : permettent de déterminer le contrôleur à appeler et avec quels arguments. Cela permet aussi d'avoir des URL plus élégantes.
- **Les services** : ont pour objectif d'organiser l'instanciation d'objets et de donner accès à une méthode n'importe où dans l'application.
- **Les événements** : ont pour vocation d'exécuter un code à un certain moment.

Nous ne présenterons pas tous ces outils car cela ne serait pas très intéressant, mais nous tenons à développer un peu plus Doctrine car c'est le composant qui nous a demandé le plus de travail. En effet, en utilisant Doctrine nous devons mettre de côté tout ce que nous avons appris jusque là sur la gestion de base de données.

3.2.1 Doctrine[3]

Avec doctrine, nous n'avons pas écrit une seule ligne de SQL, même pas pour créer les tables.

Note : en utilisant un ORM, les objets dont il s'occupe s'appellent des entités et on ne dit pas enregistrer une entité, mais persister une entité.

Créer une entité

Symfony met à disposition un générateur d'entité via la console ².

Pour chaque entité il a fallu choisir :

- Le nom de l'entité en utilisant la forme "NomBundle:NomEntité"
- Le format de configuration : nous avons choisi le format annotation car il centralise tout dans le même fichier, ce qui est donc plus simple
- Une liste de champs (Sauf id généré automatiquement), et pour chaque champ choisir :
 - Son nom
 - Son type parmi ceux proposés
 - Sa taille en octets (pour les types string)
 - S'il peut être nul ou non
 - Si sa valeur doit être unique ou non

Après ça, deux fichiers sont générés par entité. Un fichier `NomEntitéRepository` qui est la classe qui contiendra les méthodes personnelles de récupération de l'objet, et un fichier `NomEntité` ³.

Ce fichier est une classe qui contient les :

- Attributs (id et champs personnels)
- Méthodes (getters et setters)
- Commentaires (servent de documentation)
- Annotations (@ORM / permettent à doctrine de faire le lien avec la base de données)

Après avoir créé ces entités, il faut éventuellement les lier entre elles. Par exemple 1 mot ambigu à 1 auteur, 1 auteur peut avoir n mots ambigus, c'est donc une relation 1,n. Voici l'annotation nécessaire à la réalisation de cette liaison avec l'objet membre :

2. aperçu de la console en annexes p.34, fig.6

3. aperçu du fichier en annexes p.35, fig.6

```
/**
 * @ORM\ManyToOne(targetEntity="UserBundle\Entity\Membre", inversedBy="motsAmbigus")
 * @ORM\JoinColumn(nullable=false)
 */
private $auteur;
```

FIGURE 3.1 – Liaison auteur dans MotAmbigu

L'annotation est *ManyToOne* pour le 1,n. Il n'y a pas forcément besoin de l'option "*inversedBy*". Celle-ci permet juste d'avoir accès aux mots ambigus d'un membre depuis l'objet membre ce qui ne serait pas possible sans, c'est ce qu'on appelle la bidirectionnalité. On peut donc récupérer l'auteur d'un mot ambigu, mais aussi les mots ambigus d'un membre. Nous remarquons ensuite l'annotation *JoinColumn* qui précise que le champ ne peut pas être *null*, donc un mot ambigu a forcément un auteur.

1 mot ambigu peut être décrit par n gloses, 1 glose peut décrire n mots ambigus, c'est donc une relation n,n.

```
/**
 * @ORM\ManyToMany(targetEntity="Glose", inversedBy="motsAmbigus", cascade={"persist"})
 * @ORM\JoinTable(name="mot_ambigu_glose")
 */
private $gloses = array();
```

FIGURE 3.2 – Liaison gloses dans MotAmbigu

L'annotation ici est *ManyToMany* pour le n,n et on remarque que l'attribut est cette fois un tableau. Dans l'annotation le *cascade* permet de modifier l'objet *Glose* réel en cas de modification via l'objet *MotAmbigu*. Ensuite, ce n'est plus *JoinColumn* mais *JoinTable* car une relation n,n crée une table intermédiaire, mais ce n'est pas à nous de la gérer, doctrine s'occupe de tout à notre place. Le problème c'est que dans cette liaison on ne peut pas ajouter de champs puisque c'est doctrine qui s'en occupe. Il existe alors une astuce pour les relations n,n qui nécessitent d'autres attributs. Par exemple notre objet *Partie* est une liaison n,n entre *Membre* et *Phrase* mais nous avons besoin d'avoir la date, les gains etc. Nous avons donc créé une entité *Partie*, qui possède deux liaisons *ManyToOne*. Une vers *Membre*, l'autre vers *Phrase*.

Pour les getters et setters de ces nouveaux champs, Symfony nous est encore venu en aide en les générant automatiquement grâce à la console. Enfin pour effectuer les changements dans la base de données nous avons encore une fois utilisé la console. *Note* : on peut aussi afficher les requêtes SQL qui vont être exécutées avant de faire effectivement les changements.

Créer ses propres méthodes

Doctrine met à disposition plusieurs méthodes de base⁴ afin de manipuler une entité, néanmoins ces méthodes sont parfois insuffisantes pour une application. Pour certains besoins nous avons donc dû réaliser nos propres fonctions, nous avons alors utilisé les repository, classe qui concentre les méthodes personnelles. Doctrine met à disposition deux façons de faire pour créer sa requête. La première façon de faire est d'utiliser le DQL (Doctrine Query Language), qui ressemble assez au SQL (Exemple : SELECT m FROM UserBundle :Membre m, récupère tous les membres). La deuxième méthode est d'utiliser le QueryBuilder, c'est cette méthode que nous avons choisi d'utiliser car c'est la plus utilisée et la "plus objet". Voici notre méthode permettant de récupérer l'id et la valeur des gloses liées à un mot ambigu donné avec le QueryBuilder :

```
public function findGlosesValueByMotAmbiguValue($valeurMA) {
    return $this->createQueryBuilder('g')->select('g.id, g.valeur')
        ->innerJoin("g.motsAmbigus", "ma", "WITH", "ma.valeur = :valeurMA")->setParameter('valeurMA', $valeurMA)
        ->orderBy("g.valeur")
        ->getQuery()->getResult();
}
```

FIGURE 3.3 – Utilisation du QueryBuilder

Utiliser doctrine a été un grand changement pour nous qui étions habitués au SQL depuis plusieurs années. Cependant le temps que nous avons dû passer au début à apprendre son fonctionnement a été rentabilisé par la suite en nous évitant de développer beaucoup de méthodes redondantes.

4. aperçu disponible en annexes p.36, fig.6

3.2.2 Bootstrap

Bootstrap est un framework CSS développé par Twitter afin de créer des pages web adaptées aux différents types d'appareils (ordinateurs, tablettes, smartphones). Nous avons donc utilisé ce framework dans le but de gérer le design de notre jeu ⁵ qui se devait d'être jouable sur tous types d'appareils. [2]

3.3 Lancement d'une partie

Le développement de cette fonctionnalité ⁶ a commencé dès que l'on pouvait correctement enregistrer une phrase en base de données en s'assurant que les liaisons entre la phrase, les mots ambigus qu'elle contient et leurs gloses étaient correctement associées.

Afin d'éviter que les utilisateurs ne rejouent les mêmes phrases trop souvent, le lancement des phrases se base sur un algorithme dont le pseudo code est le suivant :

Algorithm 1: Lancement d'une phrase

```
Result: phrase
if utilisateur est connecté then
    | phrases = findIdPhrasesNotPlayedByMembre(user); // récupérer toutes les phrases
    |   non jouées du membre
else
    | phrases = findIdPhrasesNotPlayedByIpSince(ip, durée); // récupérer les phrases non
    |   jouées depuis 3 jours pour l'IP
end
// Si toutes les phrases ont été jouées
if phrases == null then
    | phrases = findAllIdPhrases(); // récupérer toutes les phrases
end
phrase = array_rand(phrases); // récupérer une phrase au hasard depuis la liste
précédente
return phrase;
```

3.4 Création d'une phrase

Nous avons essayé de faire en sorte que la création d'une phrase soit la plus intuitive possible pour le joueur. Par exemple, nous avons besoin d'entourer les mots ambigus de balises particulières afin de les récupérer et d'effectuer les traitements dessus, mais plutôt que d'expliquer au joueur comment les écrire nous avons mis un bouton qui ajoute les balises autour du mot sélectionné, ce qui permet aussi de limiter les erreurs. Après avoir sélectionné les mots ambigus de la phrase, celle-ci ressemble à :

"Un <amb id="1">avocat</amb> mange des <amb id="2">avocats</amb>."

Identifiant 1 Mot ambigu 1 Identifiant 2 Mot ambigu 2

FIGURE 3.4 – Exemple de phrase à la création

5. aperçu disponible en annexes p.40, fig.6

6. aperçu disponible en annexes p.39, fig.6

Après avoir indiqué pour chaque mot ambigu sa glose associée et validé la phrase, le traitement suivant est effectué :

Algorithm 2: Création d'une phrase

Result: phraseAdd

```
// Si la phrase est bien formée et que le joueur a les crédits nécessaires
if isValid(phrase) and crédits > coût then
    forall motsAmbigus do
        motAmbiguPhrase->motAmbigu = findOrCreate(motAmbigu); // Créer ou
        récupère le mot ambigu
        motAmbiguPhrase->phrase = phrase;
        phrase->motsAmbiguPhraseAdd(motAmbiguPhrase); // Ajoute le mot ambigu
        phrase à l'objet phrase
    end
    persist(phrase); // Persiste la phrase
    crédits -= coût; // Met à jour le nombre de crédits
    flush(); // Enregistre la phrase
    forall motsAmbigusPhrase do
        persist(réponse);
        persist(motAmbiguPhrase);
    end
    persist(partie);
    flush(); // Enregistre les MotsAmbigusPhrase, la partie et les réponses
else
    | return erreur;
end
return phrase;
```

Après ça la phrase n'est pas jouable pendant cinq minutes afin de laisser le temps à son créateur de la modifier en cas d'erreur.

3.5 Édition de phrase

La fonctionnalité⁷ qui permet d'éditer ou de modifier les phrases est importante car elle permet de corriger les fautes d'orthographe, voire de supprimer entièrement les phrases problématiques. Cette fonctionnalité s'est révélée plus complexe à implémenter que ce que nous pensions initialement car risquée pour l'intégrité des données déjà acquises.

Un objet Phrase est lié à un objet MotAmbigu via un objet MotAmbiguPhrase, car 1 phrase peut avoir n mots ambigus, et 1 mot ambigu peut se trouver dans n phrases. De plus, le même mot ambigu peut se retrouver plusieurs fois dans la même phrase, il est donc important d'enregistrer l'ordre de celui-ci dans l'objet MotAmbiguPhrase. L'objet Réponse possède donc un objet MotAmbiguPhrase et un objet Glose. La difficulté consistait donc à maintenir l'ensemble de ces liaisons cohérentes malgré la modification d'un des éléments.

"Un <amb id="1">avocat</amb> mange des <amb id="2">avocats</amb>."

Identifiant 1
Mot ambigu 1
Identifiant 2
Mot ambigu 2

FIGURE 3.5 – Exemple de phrase

7. aperçu disponible en annexes p.41, fig.6

Plusieurs cas de figure apparaissent alors :

- Si la modification ne porte pas sur un mot ambigu de la phrase elle est simple à réaliser. En effet les relations entre les tables ne sont pas affectées, seul le champ contenant la phrase est modifié.
- En ce qui concerne l'ajout et la suppression de mot ambigu, c'est aussi assez simple. En cas de suppression, il suffit de supprimer l'objet `MotAmbiguPhrase` correspondant et de refaire l'ordre. En cas d'ajout, il suffit de créer un nouvel objet `MotAmbiguPhrase`, de refaire l'ordre et de créer une réponse de départ pour ce nouveau mot ambigu pour le premier joueur.
- Dans le cas de la modification d'un mot ambigu existant, il suffit de mettre à jour le `MotAmbiguPhrase`. Si l'ancien et le nouveau mot sont proches (avocat -> avocats) leurs gloses seront probablement similaires et les précédentes réponses seront conservées. Sinon si la glose de la réponse n'est en fait pas liée au mot ambigu elle ne sera tout simplement pas prise en compte. Nous avons mis du temps à trouver cette solution car nous craignions que les réponses soient altérées par la modification, mais le fait qu'elles ne soient finalement pas prises en compte si le lien entre la glose et le mot ambigu n'existe pas règle le problème.

Étant donné les changements que peuvent causer ces manipulations, seuls les modérateurs peuvent les effectuer. Néanmoins les joueurs peuvent modifier leurs propres phrases à condition qu'elles aient été créées il y a moins de 5 minutes. Comme la phrase n'a jamais été jouée elle pourra être supprimée et recréée avec les corrections pour des questions de facilité.

3.6 Export des données

L'export des données et leur signification étaient des points cruciaux du projet. Au niveau de la signification des données il faut voir que sur Ambiguss on peut se retrouver face à deux types de phrases. D'abord les phrases dont le contexte aide les joueurs à lever l'ambiguïté, pour ces phrases il n'y a qu'une seule signification juste et les données récoltées devraient en toute logique indiquer que cette signification est celle choisie par la très grande majorité des joueurs. Exemple :

Ils ont utilisé un **bélier** pour enfoncer la grille du parc.

Ici le contexte permet facilement de déduire que le bélier fait référence à l'arme et non à l'animal.

Ensuite il y a les phrases qui sont réellement ambiguës dans le sens où le contexte n'est pas suffisant pour trancher entre les solutions proposées, dans ce cas on pourrait s'attendre à ce que les réponses des joueurs se répartissent entre les différentes solutions possibles. Néanmoins il est tout à fait possible qu'une des solutions soit plus populaire et c'est là tout l'intérêt du projet : en cas d'ambiguïté indécidable vers quel choix les joueurs se tournent-ils spontanément ? Exemple :

Les artistes de rue ont besoin d'une **assistance**.

Les artistes ont besoin d'un public ou d'aide ? Les deux sont possibles, mais actuellement les joueurs penchent plutôt vers la glose "**public**".

Dans les deux cas précédents nous avons supposé que les joueurs avaient en majorité réussi à trouver la ou les bonnes solutions, les données produites sont donc correctes. Mais que ce passe-t-il si la majorité des joueurs se trompent sur une phrase qui est réellement difficile, que se soit de par le vocabulaire utilisé ou par les règles lexicales qui entrent en jeu ? Dans ce cas les données produites sont fausses lexicalement mais elles ne sont pas inintéressantes pour autant car elles indiquent ce qu'un humain aurait probablement compris.

Pour exporter les données il nous fallait réfléchir à un moyen de les structurer, pour cela nous avons choisi d'utiliser le format JSON. Par exemple si l'on prend la phrase suivante :

Un **avocat** mange des **avocats**.

Les données concernant cette phrase seront exportées ainsi :

```
{
  "phrase": "Un <amb id=1>avocat</amb> mange des <amb id=2>avocats</amb>.",
  "reponse": [
    {
      "motAmbigu": "avocat", "ordre": 1, "nbRep": 31, "gloses": [
        { "valeur": "individu", "nbRep": "23" },
        { "valeur": "métier", "nbRep": "8" }
      ]
    },
    {
      "motAmbigu": "avocats", "ordre": 2, "nbRep": 30, "gloses": [
        { "valeur": "fruit", "nbRep": "30" }
      ]
    }
  ]
}
```

FIGURE 3.6 – Exemple d'export de données

En première ligne il y a la phrase, au sein de celle-ci les mots ambigus sont entourés par la balise "amb". De plus ils possèdent un identifiant qui permet de les différencier dans le cas où une phrase contiendrait plusieurs mots ambigus exactement identiques. Ensuite pour chacun des mots ambigus on affiche les gloses et le nombre de votes qu'elles ont obtenus. Seules celles qui ont obtenu au moins un vote apparaissent.

En plus de pouvoir exporter l'ensemble des phrases et des réponses, il est aussi possible de télécharger l'ensemble des gloses associées à chaque mot mot ambigu, indépendamment des phrases.

Les données récoltées par le jeu sont en accès libre et sont mises à jour quotidiennement.

3.7 Questions d'optimisation

Comme dans toute application, les questions de fluidité sont très importantes. Il est difficile de s'en apercevoir en cours de développement avec quelques centaines ou milliers de lignes en base de données, mais nous devons réfléchir au fait qu'avec le temps la base de données pourrait contenir des millions de lignes.

Nous avons donc placé des index sur tous les champs qui risquaient d'en avoir besoin de part leur possible utilisation de recherche (dans le champ where) comme par exemple les dates, les IP, etc. Les champs uniques et les clés étrangères sont indexés automatiquement.

3.8 Tests

Symfony propose deux types de tests, unitaires et fonctionnels. Les tests unitaires permettent de vérifier le fonctionnement d'une classe en testant ses méthodes. Les tests fonctionnels servent eux à vérifier la globalité d'une application, du routage aux vues.

Malheureusement nous n'avons pas eu le temps d'utiliser ces outils de tests et réalisons les tests à la main malgré les risques de régression et d'oubli que cela pouvait comporter. Il serait donc intéressant de développer ces tests afin d'éviter ce genre de problème ainsi que pour le gain de temps que cela pourrait engendrer sur le long terme.

4. Déploiement

Pour être dans des conditions plus proches de la réalité, mais aussi pour obtenir au plus vite des retours utilisateurs puisque nous utilisons la méthode de gestion de projet Agile, nous avons décidé de déployer le projet sur Internet dès le début.

Un des membres du groupe ayant déjà des connaissances dans le déploiement de site web sur Internet via un serveur personnel et un nom de domaine, c'est lui qui a été en charge de cette partie. Le serveur héberge donc le site web et la base de données.

4.1 Caractéristiques de l'hébergement

Le nom de domaine (calyx.fr) est enregistré chez OVH pour un coût annuel d'environ 8,4€ TTC. Il est possible de créer des sous-domaines sur celui-ci.

Le serveur est hébergé par FirstHeberg pour environ 1.8€ TTC/mois. Il s'agit d'un V.P.S (Virtual Private Server, serveur dédié virtuel). Il fonctionne avec Debian 8 comme système d'exploitation et dispose de 20Go d'espace disque SSD, 1Go de mémoire vive et d'un vCore (coeur virtuel).

4.2 Configuration

Pour des questions d'esthétisme et d'accessibilité nous avons créé un sous-domaine dédié au projet : ambiguss.calyx.fr. Pour cela il a été nécessaire de modifier la configuration ¹ d'Apache2 afin qu'il redirige les requêtes de ce nom de domaine sur le bon répertoire contenant le projet et aussi pour activer le module de réécriture d'URL dont Symfony se sert via les fichiers .htaccess.

Nous avons ensuite configuré des sauvegardes automatiques de la base de données MySQL du projet. Celles-ci sont réalisées quotidiennement dans la nuit pour ne pas ralentir la navigation sur le site. Après avoir été enregistrées dans une archive, elles sont envoyées par email au propriétaire du serveur afin de ne perdre aucune donnée en cas de problème sur le serveur. Il a fallu ensuite sécuriser le serveur contre les piratages, qui n'ont pas tardé, puisque quelques heures après la livraison du serveur, il y avait déjà plusieurs tentatives frauduleuses de connexion SSH et MySQL. Pour faire cela, nous avons utilisé plusieurs outils et conseils provenant de divers tutoriels, en voici quelques-uns :

- changer les ports des applications par défaut. Par exemple les ports de SSH et MySQL ne sont plus 22 et 3306 sur le serveur.
- iptables : pare-feu qui ouvre ou ferme certains ports. Nous l'avons configuré pour bloquer tous les ports et n'ouvrir ensuite que les ports dont nous avons besoin (SSH, HTTP, ...).

1. aperçu en annexes p.36, fig.6

- portsentry : bloque les scans de ports. Quand il détecte un scan de ports, il ajoute une règle au pare-feu pour bloquer l'IP.
- fail2ban : surveille les logs des applications pour y trouver des actions douteuses (échec de mot de passe, surcharge). S'il trouve quelque chose de suspect, il ajoute lui aussi une règle au pare-feu.

Bien que ce ne soit pas forcément nécessaire pour notre type d'application, mais comme c'est utilisé sur le serveur pour d'autres services, et utile pour le référencement, nous avons au départ autorisé l'utilisation de HTTP et HTTPS. Nous avons finalement généralisé l'utilisation de HTTPS. Ce protocole, qui était plutôt coûteux et difficile d'installation il y a quelques années, ne nous a rien coûté et s'est installé automatiquement grâce à Let's Encrypt². Let's Encrypt est un organisme fondé par l'ISRG (Internet Security Research Group) dont le but est de généraliser l'utilisation du protocole HTTPS. L'organisme est soutenu par de nombreux sponsors comme Mozilla, Cisco, OVH, Linux, Free...

4.3 Base de données

Afin d'avoir tous les mêmes données lors de nos développements, nous avons décidé d'utiliser directement la base de données du serveur plutôt que d'en avoir une chacun en local. Dans un premier temps, nous utilisions la même base de données pour les environnements de développement et de production, mais quand les premiers utilisateurs ont commencé à s'inscrire, nous avons créé une seconde base de données dédiée au développement, afin de ne pas impacter les données des utilisateurs en cas d'erreur de développement.

4.4 Problèmes rencontrés

Lors d'une réunion de travail à la faculté, nous nous sommes aperçus qu'il nous était impossible d'accéder au serveur. En effet, il semblerait que le proxy de la faculté bloque certains ports pour des raisons de sécurité. Ainsi, non seulement nous ne pouvions pas nous connecter au serveur en SSH, mais surtout, le site était inutilisable puisque le port de MySQL était aussi bloqué. Pour remédier à cela, il a fallu trouver des ports ouverts, notamment des ports utilisés pour d'autres services habituellement.

2. Let's Encrypt : <https://letsencrypt.org/>

5. Résultats et perspectives

5.1 Résultats et retours utilisateurs

En quelques chiffres Ambiguss c'est actuellement :

- **61 membres, 1778 visites**
- **332 phrases, 502 mots ambigus, 1278 gloses/annotations**
- **9587 réponses, 3071 parties**

Nous avons effectué la promotion du site d'abord via nos réseaux personnels, puis en écrivant des présentations dans divers forums dont la communauté pouvait être intéressée par le projet. Il y a notamment le forum de www.languefrancaise.net et celui d'Openclassroom.

Dès le début du projet nous savions que le jeu devrait être fonctionnel rapidement afin d'avoir le temps de bénéficier des remarques de nos premiers utilisateurs. Pour recueillir leurs avis, notre tuteur M. Lafourcade avait mis en place un Google doc partagé où nous pouvions répondre directement à leurs remarques et même demander leurs avis sur certains paramètres, par exemple sur le coût en crédits pour créer une phrase.

Ce contact avec les utilisateurs nous a permis de cerner efficacement quels étaient les points essentiels sur lesquels concentrer nos efforts, par exemple l'ergonomie pour les phases de jeu et de création. De plus ces retours nous ont aussi permis de détecter des fonctionnalités inutiles voir néfastes avant même de commencer à les développer, c'est par exemple le cas des modes de jeu "moyen" et "difficile" précédemment évoqués dans les choix relatifs au jeu.

Le côté négatif comme nous l'avons vu en cours de conduite de projet, c'est que nous accordions beaucoup d'importance à ces avis et que parfois nous perdions du temps sur des choses finalement pas si prioritaires que ça alors que l'on aurait pu développer d'autres fonctionnalités plus importantes. Cela dit nos choix se sont avérés payants, en effet après avoir interrogé quelques joueurs ceux-ci ont déclaré avoir apprécié le concept du jeu. Quant à nous, développeurs, nous nous sommes vite rendu compte de l'esprit de compétition qui pouvait naître entre les joueurs grâce au système de classement mis en place.

5.2 Perspectives

Le classement actuel peut être décourageant pour les nouveaux joueurs. En effet, l'écart en points avec les premiers du classement peut sembler insurmontable, notamment car les premiers joueurs ont pu jouer des phrases plusieurs fois avant la mise en place de la restriction qui n'accorde qu'une seule partie comptabilisée par joueur et qui ont donc acquis des points que les nouveaux joueurs ne pourront jamais rattraper.

Pour pallier ce genre de problème, on peut imaginer un classement hebdomadaire, en plus de celui existant, où tous les scores seront remis à zéro chaque semaine, ce qui assurera à tous de partir sur un pied d'égalité. Ainsi, un joueur, même tout nouveau, sera stimulé par

la perspective de la compétition, puisque la première place est potentiellement à sa portée chaque semaine.

En guise de récompense, chaque joueur pourrait se voir attribuer, en fin de semaine, un titre correspondant à son classement suivant le nombre de points cumulés sur cette période. Un joueur pourrait voir dans son profil combien de fois il a gagné chaque titre. Ces titres seraient de plus en plus difficiles à obtenir : chaque semaine seule une personne peut obtenir le plus prestigieux, seul 1% des joueurs obtiennent le titre 2, seul 10% obtiennent le titre 3 etc... C'est donc quelque chose de stimulant qui donnera certainement envie aux joueurs de s'investir régulièrement.

Pour motiver les joueurs sur le long terme nous avons réfléchi à un système de récompense via des succès que nous n'avons pas eu le temps de développer mais qu'il serait intéressant de réaliser afin de motiver les joueurs à jouer régulièrement. On peut par exemple imaginer des succès comme :

- Jouer au moins 15 phrases chaque jour pendant 15 jours
- Jouer 1000 phrases
- Créer 500 phrases

Bien sûr, pour ne pas être décourageants ces "succès" pourraient eux-mêmes être divisés en "sous-succès" représentant des niveaux différents. Par exemple un premier niveau bronze serait de créer 10 phrases, le niveau argent d'en créer 100, et le niveau or d'en créer 500.

Ce concept d'objectif quotidien se retrouve dans de nombreux jeux. Dans notre cas nous pourrions aussi offrir un bonus de point/crédits à un joueur qui crée une phrase par jour par exemple, avec un bonus qui pourrait croître également afin de l'encourager à faire durer cette habitude le plus longtemps possible..

6. Conclusion

Nous avons pour objectif de réaliser un jeu sérieux qui fournirait des ressources sur la désambiguïsation lexicale. En ce sens nous avons réussi car le jeu est jouable tant sur ordinateur que sur téléphone ou tablette, et les données produites par les joueurs peuvent être librement téléchargées. De plus des mécanismes de gamification tels que les classements, les likes et les crédits ont bien été implémentés, bien que pour cet aspect, de nombreuses améliorations soient possibles, comme évoqué dans les perspectives.

Au niveau technique, ce projet nous a permis de nous familiariser avec le framework Symfony, qui est aujourd’hui un des framework PHP les plus utilisés, ce qui fait de ce projet un très bon exemple à mettre dans notre CV.

Au niveau de l’organisation nous avons trouvé que le fait de travailler en Agile et donc d’être en contact direct avec des utilisateurs était très intéressant. Cependant il était parfois difficile de ne pas tomber dans le piège des nouvelles idées non prioritaires qui ralentissent le développement des fonctionnalités plus prioritaires.

Bibliographie

- [1] URL : <https://fr.wikipedia.org/wiki/Ambigu%C3%Aft%C3%A9>.
- [2] *Bootstrap · The world's most popular mobile-first and responsive front-end framework*. URL : <http://getbootstrap.com/> (visité le 22/05/2017).
- [3] *Databases and the Doctrine ORM (current)*. URL : <http://www.doctrine-project.org/> (visité le 22/05/2017).
- [4] *Développez votre site web avec le framework Symfony @OpenClassrooms*. URL : <https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony> (visité le 21/05/2017).
- [5] *Home - Twig - The flexible, fast, and secure PHP template engine*. URL : <https://twig.sensiolabs.org/> (visité le 22/05/2017).
- [6] INSTITUT DE L'AUDIOVISUEL ET DES TÉLÉCOMMUNICATIONS EN EUROPE (FRANCE). *Serious games : advergaming, edugaming, training*. French. OCLC : 470633474. Montpellier : IDATE, 2008. ISBN : 978-2-84822-169-4.
- [7] Mathieu LAFOURCADE, Nathalie LE BRUN et Alain JOUBERT. *Jeux et intelligence collective : résolution de problèmes et acquisition de données sur le Web*. English. OCLC : 909155572. 2015. ISBN : 978-1-78405-052-8.

Annexes

Exemple de génération d'entité

```
c:\wamp\www\ambiguss (master)
λ php bin\console doctrine:generate:entity

Welcome to the Doctrine2 entity generator

The Entity shortcut name: AppBundle:Test
AppBundle:Test

Determine the format to use for the mapping information.

Configuration format (yaml, xml, php, or annotation) [annotation]:

Instead of starting with a blank entity, you can add some fields now.
Note that the primary key will be added automatically (named id).

Available types: array, simple_array, json_array, object,
boolean, integer, smallint, bigint, string, text, datetime, datetimetz,
date, time, decimal, float, binary, blob, guid.

New field name (press <return> to stop adding fields): text
Field type [string]:

Field length [255]:
Is nullable [false]:

Unique [false]: true
true

New field name (press <return> to stop adding fields):

Entity generation
created .\src\AppBundle\Entity\Test.php
> Generating entity class C:\wamp\www\ambiguss\src\AppBundle\Entity\Test.php: OK!
> Generating repository class C:\wamp\www\ambiguss\src\AppBundle\Repository\TestRepository.php: OK!
Everything is OK! Now get to work :).
```

FIGURE 6.1 – Génération d'une entité via la console

Fichier de l'entité générée

```
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Test
 *
 * @ORM\Table(name="test")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\TestRepository")
 */
class Test
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="text", type="string", length=255, unique=true)
     */
    private $text;

    /**
     * Get id
     *
     * @return int
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * Set text
     *
     * @param string $text
     *
     * @return Test
     */
    public function setText($text)
    {
        $this->text = $text;

        return $this;
    }

    /**
     * Get text
     *
     * @return string
     */
    public function getText()
    {
        return $this->text;
    }
}
```




FIGURE 6.2 – Entité générée

Utilisation d'une entité

```
$repoUser = $this->getDoctrine()->getManager()->getRepository('UserBundle:Membre');
$membre = $repoUser->find(1);
$membres = $repoUser->findAll();
$membre = $repoUser->findOneBy(array('pseudo' => 'JohnDoe'));
$membres = $repoUser->findBy(array('banni' => '1'));
$membre = $repoUser->findOneByPseudo('JohnDoe');
$membres = $repoUser->findByBanni(1);

$membre->setBanni(1);

$em = $this->getDoctrine()->getManager();
$em->persist($membre);
$em->flush();
```

FIGURE 6.3 – Utilisation d'une entité

Configuration Apache2 du serveur

```
# Pour les requêtes sur le port 80 (HTTP)
<VirtualHost *:80>
    # Celles qui demandent ambiguss
    ServerName ambiguss.calyxe.fr

    # Sont redirigées sur le répertoire du projet
    DocumentRoot /var/www/Ambiguss/web

    # Pour le répertoire du projet
    <Directory "/var/www/Ambiguss/web">
        # On autorise la réécriture d'URL.
        AllowOverride All
    </Directory>
</VirtualHost>
```

Description des use cases

Cas d'utilisation	Description	Utilisateurs	Contraintes
S'inscrire via le site ou les réseaux sociaux	Afin d'accéder à d'autres fonctionnalités du jeu (points, crédits, création de phrases, classement...) l'utilisateur devra s'enregistrer dans la base de données et ce en remplissant un formulaire s'il décide de s'inscrire via le site ou se connecter à son réseau social (twitter ou facebook) sinon.	Visiteur	-Avoir une adresse mail valide et unique et avoir confirmé l'inscription (en cliquant sur le lien envoyé par le message de confirmation) si l'on s'est inscrit par mail. -Avoir un pseudo unique
S'authentifier via le site ou les réseaux sociaux	Si le visiteur est déjà inscrit sur le site, il pourra accéder à son compte en indiquant son email/pseudo ainsi que son mot de passe ou alors se connecter via son réseau social (twitter ou facebook). Il passera du statut de visiteur à celui de membre.	Visiteur	-Le pseudo et le mot de passe soient valides. -Identifiant Facebook ou twitter valide.
Récupérer mot de passe oublié	Si le visiteur est déjà inscrit mais qu'il a oublié son mot de passe, il pourra faire une demande afin de réinitialiser celui-ci en indiquant son adresse mail ou pseudo.	Visiteur	-Une adresse mail ou un pseudo valide.
Partager phrase sur réseaux sociaux	L'utilisateur à la possibilité de partager une phrase qu'il aime ou trouve intéressante sur son réseau social. Cela contribuera à faire connaître le jeu.	Visiteur, Membre	-Avoir un compte sur Facebook ou Twitter et être connecté sur celui-ci.
Contacteur administrateur	L'utilisateur peut contacter si besoin un des administrateurs du site en remplissant un formulaire.	Visiteur, Membre	-Être connecté ou avoir une adresse mail valide
Aimer phrase	Afin de pouvoir "populariser" les phrases et faire gagner des points aux créateurs, les membres auront accès au bouton "J'aime cette phrase".	Membre	
Voir les classements	Les membres auront accès à plusieurs systèmes de classements sont mis en place tels que la liste des meilleurs scores, les phrases les plus populaires...	Membre	

Cas d'utilisation	Description	Utilisateurs	Contraintes
Voir / Modifier le profil	Chaque membre du site aura un profil avec toutes les informations qu'il aura mentionné lors de son inscription (qu'il pourra modifier s'il le souhaite), le nombre de phrases, mots ambigus, annotations qu'il aura créé, le nombre de points et de phrases jouées... Ces informations pourront être consultées par les autres membres.	Membre	
Déconnexion	Dès qu'un utilisateur est authentifié il peut se déconnecter du jeu à tout moment et passer au statut de visiteur	Membre	
Modifier mot ambigu, annotation, phrase	Lorsqu'une phrase ou une annotation est signalée, elle est immédiatement affichée dans un tableau que seul les modérateurs peuvent voir. De là ils auront la possibilité d'agir en corrigeant la phrase, le mot ambigu ou l'annotation ou en les supprimant si le contenu est inapproprié.	Modérateur	
Voir statistiques	Les administrateur du jeu peuvent avoir accès à toutes sorte de statistiques concernant les parties jouées, les utilisateurs ou les visites...	Admin	
Bannir utilisateur	Il est possible pour un administrateur de bannir définitivement un utilisateur si celui-ci ne respecte pas les conditions générales du jeu.	Admin	
Envoyer <i>newsletter</i>	Lors de son inscription un utilisateur peut accepter ou non de recevoir une <i>newsletter</i> . Celle-ci contiendra entre autre les nouveaux scores et les nouvelles fonctionnalités du jeu s'il y en a.	Admin	
Nommer des modérateurs	L'administrateur peut attribuer le rôle de modérateur aux joueurs les plus sérieux (avec leur accords) afin qu'il puisse contribuer à garantir la "je n'arrive pas à trouver le mot" des données récoltées.	Admin	

Captures d'écrans de l'application

Jouer une phrase sur le site



FIGURE 6.4 – Jouer une phrase



FIGURE 6.5 – Résultat

Jouer une phrase sur mobile

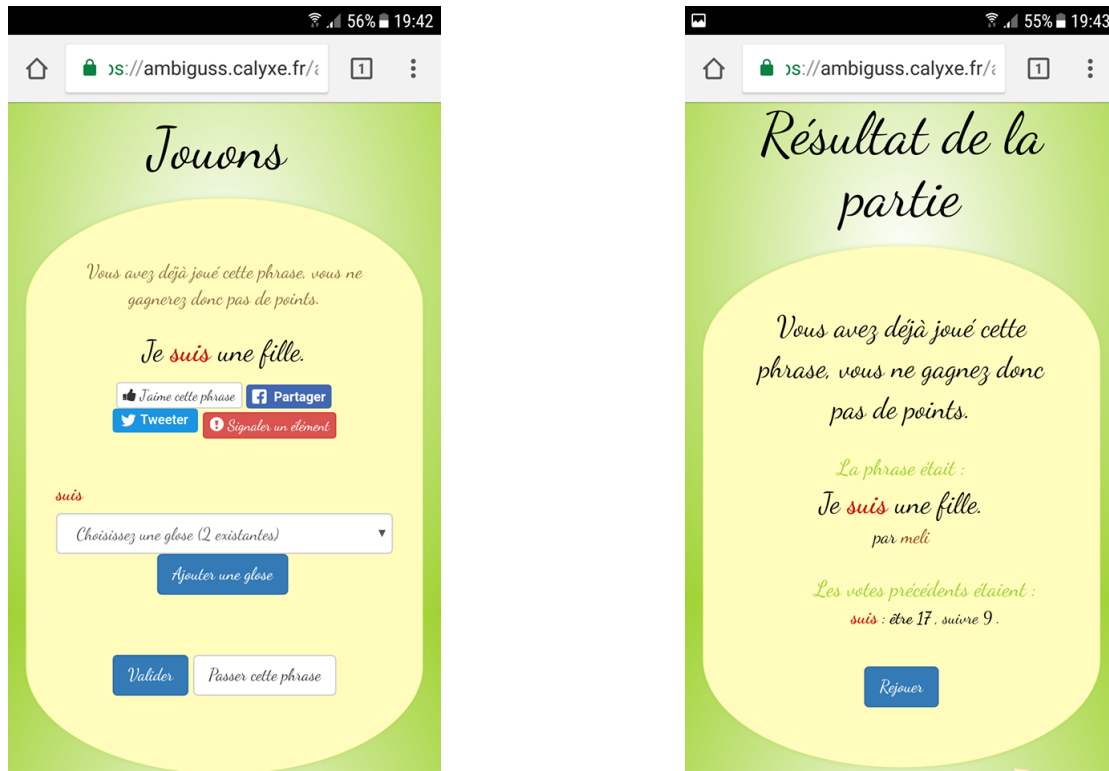


FIGURE 6.6 – Jeu sur mobile

Créer une phrase

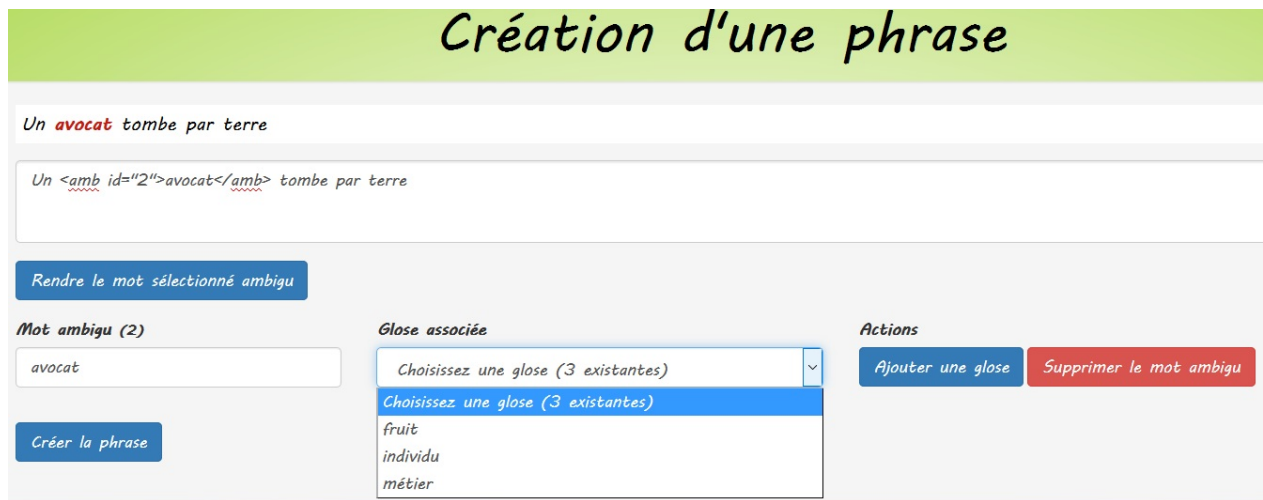


FIGURE 6.7 – Création d'une phrase

Éditer une phrase

Édition d'une phrase

Phrase originale :

La lune à son dernier quartier baignait le quartier d'une clarté insolite.

La lune à son dernier quartier baignait le quartier d'une clarté insolite.

La lune à son dernier <amb id="1">quartier</amb> <amb id="2">baignait</amb> le quartier d'une <amb id="3">clarté</amb> insolite.

Rendre le mot sélectionné ambigu

Mot ambigu (1)	Glose associée	Actions
quartier	<input type="text" value="lune"/>	Ajouter une glose Supprimer le mot ambigu
Mot ambigu (2)	Glose associée	Actions
baignait	<input type="text" value="envelopper"/>	Ajouter une glose Supprimer le mot ambigu
Mot ambigu (3)	Glose associée	Actions
clarté	<input type="text" value="luminosité"/>	Ajouter une glose Supprimer le mot ambigu

Signalé

☐ Oui

☒ Non

[Voir/cacher les jugements en cours](#)

Autres

Manque un MA (quartier)

Par alex, le 08/05/2017 06:23

[Signalement valide](#) [Signalement invalide](#)

FIGURE 6.8 – Page d'édition de phrases

Classement des joueurs

Classement général

Afficher 10 éléments Rechercher :

#	Pseudo	Titre	Inscription	Points	Nombre de phrases	Nombre de likes
1	alex	Ambigator	23/02/2017	83108	36	32
2	jcc	Grand Maître	07/05/2017	64353	0	0
3	nat	Grand Maître	28/03/2017	64252	110	106
4	kaput	Grand Maître	28/03/2017	64198	61	50
5	meli	Cavalier	02/03/2017	55216	6	4
6	Oldchimera	Cavalier	20/05/2017	54173	0	0
7	Mhcty52	Cavalier	20/05/2017	52948	0	0
8	nills	Cavalier	04/04/2017	50576	47	51
9	mlf	Cavalier	02/04/2017	48612	28	33
10	victor	Cavalier	17/04/2017	22870	0	0

Affichage de l'élément 1 à 10 sur 58 éléments

Précédent 1 2 3 4 5 6 Suivant

FIGURE 6.9 – Classement des joueurs par points